NFAs in OCAML

We describe here our implementation of NFAs in OCAML. The files that implement this are in the ocaml folder, namely nfa.ml and nfa.mli.

The interface is not all that dissimilar:

```
module type NFA =
  sig
    module A : Alphabet.A
    type state = int
    type elem
    type str

    module D : Dfa.DFA with module A = A
                        and type state = state
                        and type elem = elem
                        and type str = str

    type trans = state -> elem -> state list
    type eps_trans = state -> state list
    (* The nfa type *)
    type t

    val make : int -> eps_trans -> trans -> state list -> t

    val epsilonClose : t -> state list -> state list
    val delta : t -> state list -> elem -> state list
    val deltaStar : t -> state list -> str -> state list
    val accept : t -> str -> bool
    val isFinal : t -> state -> bool

    (* Returns the accepted strings of at most given length *)
    val acceptedStrings : t -> int -> str list

    val union : t -> t -> t
    val complement : t -> t
    val concatenate : t -> t -> t
    val star : t -> t
  end

module Make(A : Alphabet.A) : NFA with type elem = A.elem
                                   and type str = A.t
```

The first thing that stands out is the use of a "submodule" D to represent DFAs on the same alphabet. As every DFA can be considered as a corresponding NFA, this would be handy to have. (In fact we have at the moment omitted a way to obtain an NFA from a DFA, but it could be easily added).

The big change is of course in the transition functions. We separate the "normal" transitions, which are implemented as functions state -> elem -> state list from the "epsilon transitions", which are implemented as function state -> state list. The definition of an NFA via the make function asks for both. The only other real addition is an epsilonClose function, that takes a set of states and computes the epsilon closure of that set.

TODO